

Scheduling of Event-Triggered Networked Control Systems using Timed Game Automata

Dieky Adzkiya¹ and Manuel Mazo, Jr.²

¹Department of Mathematics, Institut Teknologi Sepuluh Nopember, Indonesia. Part of this work was done in the Delft Center for Systems and Control, TU Delft, The Netherlands.
(e-mail: dieky@matematika.its.ac.id)

²Delft Center for Systems and Control, TU Delft - Delft University of Technology, The Netherlands. (e-mail: m.mazo@tudelft.nl).

Abstract

We discuss the scheduling of a set of networked control systems implemented over a shared communication network. Each control loop is described by a linear-time-invariant (LTI) system with an event-triggered implementation. We assume the network can be used by at most one control loop at any time instant and after each controller update, a pre-defined channel occupancy time elapses before the network is available. In our framework we offer the scheduler two options to avoid conflicts: using the event-triggering mechanism, where the scheduler can choose the triggering coefficient; or forcing controller updates at an earlier pre-defined time. Our objective is avoiding communication conflict while guaranteeing stability of all control loops. We formulate the original scheduling problem as a control synthesis problem over a network of timed game automata (NTGA) with a safety objective. The NTGA is obtained by taking the parallel composition of the timed game automata (TGA) associated with the network and with all control loops. The construction of TGA associated with control loops leverages recent results on the abstraction of timing models of event-triggered LTI systems. In our problem, the safety objective is to avoid that update requests from a control loop happen while the network is in use by another task. We showcase the results in some examples.

1 Introduction

Networked control systems (NCSs) are spatially distributed systems in which the communication between sensors, actuators and controllers occurs through a shared band-limited digital communication network, as shown in Fig. 1. Such

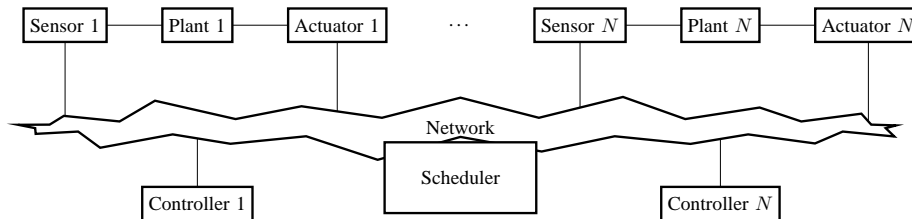


Figure 1: Topology of a set of N networked control systems.

structures bring many advantages, for instance reduced wiring and maintenance costs as well as an increased flexibility and reconfigurability. NCSs occur in numerous applications, including power systems [1], aircrafts and automobiles [2], and process control [3].

Notice that NCSs are implemented over shared communication resources, most often over digital channels. The impact of such communication infrastructures on control systems has been studied in the last decade [4, 5, 6, 7]. In particular the applicability of wireless communications in NCSs has been discussed in [8, 9, 10] among others. The delays introduced by these shared resources on the feedback loops are critical to guarantee stability and performance. Furthermore, when several control loops are implemented over a shared communication channel, bandwidth becomes a scarce resource, the usage of which needs to be minimized by each controller.

For these reasons, the traditional time-triggered controller implementations, i.e. based on periodic sampling, are not suitable anymore. With the objective of minimizing the bandwidth usage, event-based approaches resulting in aperiodic controller updates have been proposed in [11, 12, 13, 14, 15, 16]. These aperiodic paradigms introduce a new challenge in the design: the scheduling of transmissions.

A communication network has finitely many channels. The number of channels represents the maximum number of messages that can be sent simultaneously over the network. If the number of channels equals the number of control loops, we do not need any scheduler because every control loop has its own communication channel. However in practice, the number of channels is smaller than the number of control loops. Thus we need a scheduler to decide which control loop has access to the network at any time instant while guaranteeing stability of all control loops. Additionally the scheduler can optimize a certain combination of control performance and bandwidth usage. In the context of periodic traffic sources, or aperiodic with known deadlines a-priori, there are well-studied scheduling techniques that are capable of guaranteeing certain delay bounds for the traffic [17, 18, 19]. In the event-based context, i.e. sporadic traffic with unknown deadlines, the problem has been less studied and becomes more challenging [20, 21, 22, 23]. In [21, 22, 23], the authors propose a joint design (codesign) of a control law and a scheduling law for several types of NCSs. Although the co-design strategy can improve the control performance signifi-

cantly, if a new control loop is introduced to the NCSs, the whole co-design procedure has to be executed again, which can be extremely time consuming. In order to mitigate this issue, we propose an approach that separates the design of controllers and schedulers.

In this paper, we design a scheduler for a set of NCSs over a shared communication network (cf. Fig. 1). Our objective is avoiding communication conflict while guaranteeing stability of all NCSs. Each NCS is a linear-time-invariant (LTI) system. The controllers are implemented in an event-triggered fashion where the delays between reading the state and updating the actuators are ignored [12]. With respect to the shared communication network, we assume it has a single communication channel. Furthermore after each controller update, a pre-defined channel occupancy time elapses before the network is available. We consider schedulers that after each transmission of measurements, decide the policy for the next update. These policies can be to either let the next update be decided based on a triggering mechanism (to be chosen among a set of them guaranteeing different performances) or forced to be at an earlier pre-defined time. If we do not allow the scheduler to force earlier controller updates, the bandwidth usage is decreased at the cost of worse control performance (or slower convergence). On the other hand if we allow the scheduler to force earlier controller updates, we obtain a better control performance (or faster convergence) at the cost of increased bandwidth usage. In this case, nothing prevents the scheduler from always using earlier updates and never use the event-triggering mechanism. This may result in an undesired over-use of the communication channel, and could be prevented by introducing costs to the model, which would result in priced timed game automata (PTGA). Unfortunately to the best of our knowledge, no results are available in the literature allowing the synthesis of strategies over PTGA with safety objectives. Only the synthesis of strategies over PTGA with a reachability objective are available [24, 25]. Thus we propose an alternative approach to prevent the undesired schedule by limiting the consecutive earlier updates.

The scheduling problem can be formulated as a timed safety game: given a model and a set of bad states, we seek to construct a strategy such that the model supervised by the strategy constantly avoids the bad states. In our problem, the safety problem at hand is to avoid that update requests from a control loop happen while the network is in use by another task. We focus our attention to the design of a scheduler by leveraging techniques originally developed for network of timed game automata (NTGA) [26]. An NTGA is the parallel composition of timed game automata (TGA), which are timed safety automata (TSA) in with the set of actions is partitioned into controllable and uncontrollable actions. We choose NTGA modeling framework because of the following two reasons. First of all, it allows us to extend the methods in [27]. The authors of [27] discuss formal abstraction of the timing behavior of LTI systems with event-triggered implementation as TSA. The second reason is that the solution of timed safety game over NTGA can be computed by using backward algorithms [28, 26] or on-the-fly algorithms [29]. Moreover the algorithms have been implemented in some freely available software tools [30, 29]. The

procedure to generate the scheduler (or the strategy) is as follows. First, we construct an NTGA from a set of NCSs. The NTGA is obtained by taking the parallel composition of the TGA associated with the network and with all control loops. Then we characterize the bad states, i.e. states corresponding to a communication conflict. Finally the scheduler is defined as the solution of timed safety game over the NTGA.

Timed automata (TA) [31] are a general modeling framework for a wide range of real-time systems, such as in web services [32], audio/video protocols [33], bounded retransmission protocols [34], collision avoidance protocols [35, 36] and commercial field bus protocols [37]. Timed safety automata (TSA) [38] are a simplified version of TA. In order to enforce progress properties, TSA use local invariant conditions whereas TA use Büchi or Muller accepting conditions. The scheduling problem over TA and its variants has been already studied in the literature e.g. applied to the scheduling of a steel plant, a job shop and a task graph in [39], [40], and [41], respectively. Furthermore the optimal scheduling of a production w.r.t. a predefined cost for a finite time horizon has been investigated in [42, 43]. In this case, the models are TA with weights (or costs) on both locations and edges, so called priced timed automata in [44] and weighted timed automata in [45]. Finally the optimal scheduling for infinite time horizon is discussed in [46].

The rest of this manuscript is structured as follows. Section 2 recalls some modeling frameworks and preliminaries. Section 3 proposes a procedure to synthesize a conflict-free scheduler. Two experimental results are discussed in Section 4. The first experiment allows the scheduler to force earlier controller updates where the number of consecutive earlier updates is limited to 4. The second experiment does not allow the scheduler to force earlier controller updates. In this experiment, the scheduler can choose the triggering coefficient among three choices. Finally the conclusions and possible future research directions are summarized in Section 5.

2 Models and Preliminaries

2.1 Timed Automata

A timed automaton (TA) [31] is a finite automaton (namely, a directed graph containing finitely many nodes and finitely many labeled edges) extended with real-valued variables, which is usually employed to model real-time systems. The real-valued variables model logical clocks, that are initialized to zero when the system is started and thereafter increase synchronously at the same rate. We shall refer to these variables as simply “clocks”. Clock constraints are used to restrict the behavior of the automaton. An edge transition can be taken when the edge is enabled. Edges are enabled if the values of the clocks satisfy the guard conditions associated with the edge. Additionally, some clocks may be reset to zero when an edge is taken. Originally, Büchi and Muller accepting conditions are used to enforce progress properties [31]. A simplified version called

timed safety automata [38] uses local invariant conditions to specify progress properties. In this work, we focus on timed safety automata and refer them as timed automata for simplicity.

We define C as the set of finitely many clocks, Act as the set of finitely many actions and \mathbb{N}_0 as the set of natural numbers including zero $\{0, 1, \dots\}$. A clock constraint is a conjunctive formula of atomic constraints of the form $x \bowtie n$ or $x - y \bowtie n$ for $x, y \in C$, $\bowtie \in \{\leq, <, =, >, \geq\}$ and $n \in \mathbb{N}_0$. Clock constraints will be used as guards on edges and location invariants. We use $\mathcal{B}(C)$ to denote the set of clock constraints.

Definition 1 (Timed Automaton). *A timed automaton TA is a sextuple $(L, \ell_0, \text{Act}, C, E, \text{Inv})$ where*

- L is a set of finitely many locations (or nodes);
- $\ell_0 \in L$ is an initial location;
- Act is a set of finitely many actions;
- C is a set of finitely many real-valued clocks;
- $E \subseteq L \times \mathcal{B}(C) \times \text{Act} \times 2^C \times L$ is a set of edges;
- $\text{Inv} : L \rightarrow \mathcal{B}(C)$ assigns invariants to locations.¹

Location invariants are restricted to constraints that are downwards closed, in the form: $c \leq n$ or $c < n$ where c is a clock and $n \in \mathbb{N}_0$.

Sometimes we write $\ell \xrightarrow{g, a, \mathbf{r}} \ell'$ when $(\ell, g, a, \mathbf{r}, \ell') \in E$. Furthermore we write $\ell \longrightarrow \ell'$ to denote the existence of an edge from ℓ to ℓ' with arbitrary labels.

The semantics of a TA are defined as a transition system where a state consists of the current location and the current value of clocks. There are two types of transitions between states depending on whether the automaton: delays for some time (a delay transition), or takes an enabled edge (a discrete transition).

To keep track of clock values, we use functions known as clock assignments $u : C \rightarrow \mathbb{R}_{\geq 0}$ and we employ $u \models g$ (u satisfies g) to denote that the clock values of u satisfy the guard g . For $d \in \mathbb{R}_{\geq 0}$, let $u + d$ denote the clock assignment that maps all $c \in C$ to $u(c) + d$. For a set of clocks $\mathbf{c} \subseteq C$, let $u[\mathbf{c}]$ denote the clock assignment that maps all clocks in \mathbf{c} to 0 and agrees with u for the rest of clocks in $C \setminus \mathbf{c}$.

Definition 2 (Operational Semantics). *The semantics of a timed automaton is a transition system (also known as a timed transition system) in which states are pairs of location ℓ and clock assignment u , and transitions are defined by the rules:*

- *Delay transition:* $(\ell, u) \xrightarrow[\text{TS}]{d} (\ell, u + d)$ if $u \models \text{Inv}(\ell)$ and $(u + d) \models \text{Inv}(\ell)$ for a non-negative real number $d \in \mathbb{R}_{\geq 0}$;

¹Recall that 2^C denotes the power set of C .

- *Discrete transition:* $(\ell, u) \xrightarrow[TS]{a} (\ell', u')$ if $\ell \xrightarrow{g, a, \mathbf{r}} \ell'$, $u \models g$, $u' = u[\mathbf{r}]$ and $u' \models \text{Inv}(\ell')$.

A run of a timed automaton is a sequence of alternating delay and discrete transitions in the transition system.

We denote by $\text{Runs}(\text{TA})$ the set of runs of timed automaton TA starting from the initial state (ℓ_0, u_0) where u_0 is a clock assignment that maps all $c \in C$ to 0. Additionally, if ρ is a finite run, the last state of the run is denoted by $\text{last}(\rho)$.

The set of actions Act (cf. Definition 1) is assumed to consists of symbols for input actions $a?$, output actions $a!$ and internal actions $*$. Synchronous communication between different TA is done by hand-shake synchronization using input and output actions.

To model concurrent systems, several TAs can be extended with parallel composition that takes into account the synchronous communication. Parallel composition of TAs is also called network of timed automata (NTA). Essentially the parallel composition of a set of TAs is the product of the TAs. Building the product timed automaton is an entirely syntactical but computationally expensive operation. The reader is referred to [47, Sec. 5] for an example on the composition of two TAs.

The semantics of an NTA are defined as a transition system where a state consists of a vector of current locations and the current value of clocks in all TAs [48].

2.2 Timed Game Automata

A timed game automaton is a timed automaton in which the set of actions is partitioned into controllable and uncontrollable actions. The former are actions that can be triggered by the controller, whereas the latter only by the environment/opponent.

Definition 3 (Timed Game Automaton). *A timed game automaton TGA is a septuple $(L, \ell_0, \text{Act}_c, \text{Act}_u, C, E, \text{Inv})$ where*

- $(L, \ell_0, \text{Act}_c \cup \text{Act}_u, C, E, \text{Inv})$ is a timed automaton;
- Act_c is a set of controllable actions;
- Act_u is a set of uncontrollable actions;
- $\text{Act}_c \cap \text{Act}_u = \emptyset$.

Similar to TA, TGA can also be extended with parallel composition (essentially the synchronized cartesian product of TGA). The parallel composition of TGAs is called a “network of timed game automata” (NTGA) which is formally defined as:

Definition 4 (Parallel Composition). Let $\text{TGA}^i = (L^i, \ell_0^i, \text{Act}_c^i, \text{Act}_u^i, C^i, E^i, \text{Inv}^i)$ be a timed game automaton for $i \in \{1, \dots, n\}$. The parallel composition of $\text{TGA}_1, \dots, \text{TGA}_n$ denoted by $\text{TGA}_1 \mid \dots \mid \text{TGA}_n$ is a timed game automaton $\text{TGA} = (L, \ell_0, \text{Act}_c, \text{Act}_u, C, E, \text{Inv})$ where

- $L = L^1 \times \dots \times L^n$;
- $\ell_0 = (\ell_0^1, \dots, \ell_0^n)$;
- $\text{Act}_c = \{*\} \cup \bigcup_{i=1}^n \{a \in \text{Act}_c^i \mid a \text{ is an internal action}\}$;
- $\text{Act}_u = \{\oplus\} \cup \bigcup_{i=1}^n \{a \in \text{Act}_u^i \mid a \text{ is an internal action}\}$;
- $C = C^1 \cup \dots \cup C^n$;
- E is defined according to the following two rules:
 - a TA makes a move on its own via its internal action: the edge is controllable iff the internal action is controllable;
 - two TAs move simultaneously via a synchronizing action: the edge is controllable iff both input and output actions are controllable (i.e. the environment has priority over the controller);
- $\text{Inv}((\ell_1, \dots, \ell_n)) = \text{Inv}^1(\ell_1) \wedge \dots \wedge \text{Inv}^n(\ell_n)$.

In the parallel composition of TGAs, a pair of input and output actions is denoted as a single action. Thus the sets Act_c and Act_u do not contain any input and output actions. A synchronizing action should be defined as an element of Act_c if it is controllable and an element of Act_u if it is not controllable. In Definition 4, let us remark that both Act_c and Act_u do not contain synchronizing actions for simplicity. Any controllable synchronizing action is denoted by $*$, whereas any uncontrollable synchronizing action is denoted by \oplus .

Given an NTGA, we are interested in solving the following safety objective: is it possible to find a strategy for the triggering of controllable actions guaranteeing that a set of pre-specified bad states are never reached regardless of what and when uncontrollable actions take place? More formally given an NTGA and a set of bad states \mathcal{A} , we seek to construct a strategy f such that the NTGA supervised by f constantly avoids \mathcal{A} .

A strategy [26] is a function that during the course of a game constantly gives information about what the controller should do in order to win the game. At any given situation, the strategy could suggest the controller to either “take a particular controllable action” or “do nothing at this point in time”, i.e. delay, which will be denoted by the symbol (controllable action) λ .

Definition 5 (Strategy [29, Definition 3]). Let $\text{TGA} = (L, \ell_0, \text{Act}_c, \text{Act}_u, C, E, \text{Inv})$ be a timed game automaton. We define $\text{TA} = (L, \ell_0, \text{Act}_c \cup \text{Act}_u, C, E, \text{Inv})$ as the timed automaton derived from the timed game automaton. A strategy f over TGA is a partial function from $\text{Runs}(\text{TA})$ to $\text{Act}_c \cup \{\lambda\}$ s.t. for every finite run ρ , if $f(\rho) \in \text{Act}_c$ then $\text{last}(\rho) \xrightarrow[\text{TS}]{f(\rho)} (\ell', u')$ for some (ℓ', u') .

A strategy f over TGA is called state-based or memoryless whenever $last(\rho) = last(\rho')$ implies $f(\rho) = f(\rho')$, for each $\rho, \rho' \in \text{Runs}(\text{TA})$. The restricted behavior of an NTGA controlled with some strategy f is defined by the notion of outcome [28].

A strategy f is winning from a state if all maximal runs [29, p. 70] in the outcome originated from that state are winning. A state is winning if there exists a winning strategy f from that state. The winning states can be computed by using backward algorithms [28, 26] or on-the-fly algorithms [29]. Software tools are also available that solve safety control problems, e.g. the implementation from Verimag [30] or UPPAAL-Tiga [29], which implement the backward and on-the-fly algorithms respectively.

2.3 Event Triggered Control Systems

We consider linear-time-invariant (LTI) systems of the form

$$\dot{\xi}(t) = A\xi(t) + Bv(t), \quad \xi(t) \in \mathbb{R}^n, \quad v(t) \in \mathbb{R}^m \quad (1)$$

where A and B are matrices of appropriate dimensions. We assume the existence of linear state-feedback laws $v(t) = K\xi(t)$ rendering the closed-loop system globally asymptotically stable, where K is a matrix of appropriate dimensions.

Assume a sample-and-hold implementation of the control law is in place keeping the input signal constant between update times, i.e.

$$v(t) = K\xi(t_k), \quad t \in [t_k, t_{k+1}[\quad (2)$$

where t_0, t_1, \dots is a divergent sequence of update times. For simplicity of presentation, we ignore the presence of delays between reading the state and updating the actuators. The interested reader is referred to [12] for more details, including accounting for delays.

In event-triggered implementations, the sequence of update times is decided on run-time based on the state of the plant [12]. Let $\xi(t)$ represent the solution of (1)-(2). We define an auxiliary variable $e(t)$ representing the difference between the sampled state $\xi(t_k)$ and the current state $\xi(t)$ of the system:

$$e(t) = \xi(t_k) - \xi(t), \quad t \in [t_k, t_{k+1}[\quad k \in \mathbb{N}_0.$$

The event-triggering approach in [12], proposes the following sampling triggering law:

$$t_{k+1} = \min\{t \mid t > t_k \text{ and } |e(t)|^2 \geq \sigma|\xi(t)|^2\}, \quad (3)$$

where $\sigma \in]0, \bar{\sigma}[\subset \mathbb{R}^+$ is the triggering coefficient, which establishes a trade off between quality of control (convergence rate to the equilibrium) and the amount of transmissions triggered. The inter-sample time of the state x , denoted by $\tau_\sigma(x)$, is defined as the time between consecutive updates when the sampled state is x :

$$\tau_\sigma(x) = \min\{t \mid |e(t)|^2 \geq \sigma|\xi(t)|^2 \text{ and } \xi(0) = x\}. \quad (4)$$

2.4 Abstraction of Event Triggered Control Systems as Timed Automata

In [27], the authors propose an approach to characterize the sampling behavior of LTI systems with event-triggered implementation as TAs. The approach abstracts the spatial and temporal dependencies of the original system. The following definitions summarize the approach.

Definition 6 (Flow Pipe). *The set of reachable states or the flow pipe at the time interval $[t_1, t_2]$ from a set of initial states X_0 is denoted by*

$$\mathcal{X}_{[t_1, t_2]}(X_0) = \bigcup_{t \in [t_1, t_2]} \{\xi(t) \mid \xi(0) \in X_0\}.$$

Definition 7 ([27]). *A timed automaton abstracting the triggering timing behavior of system (1)-(2) with triggering coefficient σ is given by*

$\text{TA}^\sigma = (L^\sigma, \ell_0^\sigma, \text{Act}^\sigma, C^\sigma, E^\sigma, \text{Inv}^\sigma)$ *where*

- $L^\sigma = \{\mathcal{R}_1^\sigma, \dots, \mathcal{R}_q^\sigma\};$
- $\ell_0^\sigma = \mathcal{R}_s^\sigma$ *such that* $\xi(0) \in \mathcal{R}_s^\sigma;$
- $\text{Act}^\sigma = \{*\};$
- $C^\sigma = \{c\};$
- $(\mathcal{R}_s^\sigma, \underline{\tau}_s^\sigma \leq c \leq \bar{\tau}_s^\sigma, *, \{c\}, \mathcal{R}_t^\sigma) \in E^\sigma$ *if* $\mathcal{X}_{[\underline{\tau}_s^\sigma, \bar{\tau}_s^\sigma]}(\mathcal{R}_s^\sigma) \cap \mathcal{R}_t^\sigma \neq \emptyset;$
- $\text{Inv}^\sigma(\mathcal{R}_s^\sigma) = \{c \mid 0 \leq c \leq \bar{\tau}_s^\sigma\}$ *for all* $s \in \{1, \dots, q\}.$

Each location \mathcal{R}_s^σ of this TA, is associated with a set of possible states x of the system (1). We abuse slightly notation denoting both the location and the associated region with the same symbol \mathcal{R}_s^σ . The suggestion in [27] is to partition the state space of the control system in conic regions pointed at the origin, each of which would be associated to a location of the TA. The TA has one clock variable c that represents the time elapsed since the last update. According to [12], given a fixed sampled state, the inter-sample time is uniquely defined, i.e. it is deterministic. In general, when the sampled state is different, the inter-sample time is also different. The notation $\underline{\tau}_s^\sigma$ and $\bar{\tau}_s^\sigma$ represents the lower and upper bounds of the inter-sample time for sampled states in \mathcal{R}_s^σ . In [27] it is shown formally that the TA abstracts the timing behavior of the event-triggered system, implying that:

$$\forall s \in \{1, \dots, q\}, \forall x \in \mathcal{R}_s^\sigma : \tau_\sigma(x) \in [\underline{\tau}_s^\sigma, \bar{\tau}_s^\sigma].$$

Remark 1. *In principle it could happen that $\tau_\sigma(x) = \infty$ for some states x of the system. In practice, one would always impose a maximum time between transmissions to maintain a minimum level of feedback. This practical solution is suggested in [27] to guarantee having always $\bar{\tau}_s^\sigma < \infty$, as otherwise the TA model would become useless for scheduling purposes.*

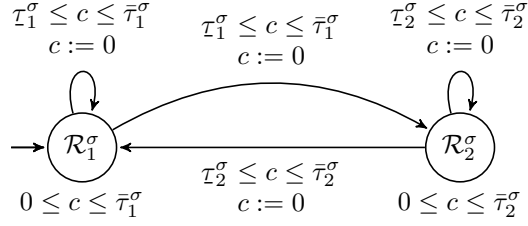


Figure 2: A timed automaton modeling a control loop with event-triggered implementation. The unsourced arrow indicates the initial location. The internal action $*$ is omitted in the figure.

From Definition 7 it is also trivial to see that if $\xi(t) = x \in \mathcal{R}_s^\sigma$ then $\xi(t + \tau_\sigma(x)) \in \mathcal{R}_j^\sigma$, with \mathcal{R}_j^σ being one of the end locations in the set of edges with starting location \mathcal{R}_s^σ . The outgoing edges of \mathcal{R}_s^σ are enabled if the time elapsed since the last update is between τ_s^σ and $\bar{\tau}_s^\sigma$. Only one action denoted by $*$ is present in this model, and since taking any edge is interpreted as updating the input value, all edges are labeled with action $*$ and reset the clock variable. Note that the system may remain in location \mathcal{R}_s^σ for at most $\bar{\tau}_s^\sigma$ time units, as a triggering event is guaranteed to happen before that instant. A graphical representation of a simple TA of the form of those from Definition 7 is shown in Fig. 2.

3 Scheduling of Event-Triggered Control Systems

Consider a set of event-triggered networked control systems (NCSs) sharing a common communication channel (cf. Fig. 1). Each control loop consists of a sensor, a plant, an actuator, and a controller, interconnected through the shared communication network. Assume that the network can be used by at most one control loop at any time instant. If several control loops request access to the channel while the network is in use a conflict arises, and at most one control loop will be chosen nondeterministically to access the network. While in time-triggered control systems these type of problems can be prevented by appropriate scheduling, when one or several control-loops are event-triggered a-priori scheduling is a much more challenging task because of the unknown update times.

In this section, we propose an approach based on NTGA to avoid such conflicts. We consider schedulers that after each update of a control loop (transmission of measurements, computation of control and transmission of actuation signal to actuators) decide whether the next update time of each control loop should:

- be based on a triggering mechanism selected from a set of finitely many triggering coefficients $\{\sigma_1, \dots, \sigma_p\}$; or

- forced to be at a pre-defined time (earlier than the minimum expected inter-sample time).

We synthesize scheduling strategies by: constructing an NTGA from a set of NCSs (cf. Section 3.1), characterizing the bad states, i.e. states corresponding to a communication conflict (cf. Section 3.2), and finally synthesizing a supervising strategy ensuring that the NTGA avoids the bad states.

3.1 Model: Network of Timed Game Automata

In what follows, we describe the procedure employed to construct an NTGA from a given set of event-based NCSs. We start constructing a TGA associated with the shared communication network. Next, for each event-triggered control loop, we generate a TGA as a modification of the TA described in Section 2.4. Finally, the NTGA is obtained by taking the parallel composition of the TGA associated with the network and with all control loops.

3.1.1 Communication Network

Denote the TGA corresponding to the shared communication network by TGA^{net} (cf. Definition 8). TGA^{net} has three locations *Idle*, *InUse* and *Bad*, where the initial location is *Idle* (cf. Fig. 3). The location *Idle* represents the network being available, *InUse* represents the network being used by a control loop and *Bad* represents a conflict occurred. The active location changes from *Idle* to *InUse* when a control loop requests access to the channel to perform an update, which also forces the reset of the clock variable c . The channel is occupied for Δ time units before the network is freed again to service the control tasks. During this time, the active location is *InUse*, and after that time the active location changes to *Idle*. When the active location is *InUse* and another control loop requests access then the active location changes to *Bad*. Once the network enters the location *Bad*, the network cannot leave the location, i.e. *Bad* is an absorbing location. Notice that this is a somewhat conservative model, as we consider every control loop occupies the channel the whole time Δ . One could trivially adjust this simple model, and the subsequent work, to associate different occupancy times to different control loops.

Definition 8. Let Δ represent the maximum channel occupancy time, a timed game automaton associated with the communication network is given by $\text{TGA}^{net} = (L^{net}, \ell_0^{net}, \text{Act}_c^{net}, \text{Act}_u^{net}, C^{net}, E^{net}, \text{Inv}^{net})$ where

- $L^{net} = \{\text{Idle}, \text{InUse}, \text{Bad}\};$
- $\ell_0^{net} = \text{Idle};$
- $\text{Act}_c^{net} = \{*\};$
- $\text{Act}_u^{net} = \{up?\};$
- $C^{net} = \{c\};$

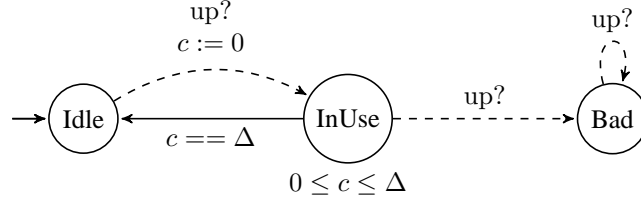


Figure 3: A timed game automaton modeling the shared communication network. The solid and dashed arrows represent controllable and uncontrollable edges, respectively.

- $E^{net} = \{(Idle, true, up?, \{c\}, InUse), (InUse, c = \Delta, *, \emptyset, Idle), (InUse, true, up?, \emptyset, Bad), (Bad, true, up?, \emptyset, Bad)\};$
- $Inv^{net}(InUse) = \{c \mid 0 \leq c \leq \Delta\},$
 $Inv^{net}(Idle) = \{c \mid c \geq 0\}, Inv^{net}(Bad) = \{c \mid c \geq 0\}.$

The guard *true* represents a condition that is always satisfied, for example $c \geq 0$.

3.1.2 Control Loops

Given a control loop, we construct the timed game automata TGA^{cl} allowing a supervisor (scheduler) to either: force earlier controller updates than those dictated by the event-triggering mechanism, or choose a triggering coefficient for the event-triggering mechanism.

Definition 9. Consider a set of timed automata $TA^{\sigma_j} = (L^{\sigma_j}, \ell_0^{\sigma_j}, Act^{\sigma_j}, C^{\sigma_j}, E^{\sigma_j}, Inv^{\sigma_j})$ generated from an event-triggered control loop with triggering coefficient $\sigma_j \in]0, \bar{\sigma}[$ for $j \in \{1, \dots, p\}$ and assume that $\mathcal{R}_s^{\sigma_1} = \dots = \mathcal{R}_s^{\sigma_p}$ for all $s \in \{1, \dots, q\}$. Consider also a set of earlier update time parameters $\{\underline{d}_1, \bar{d}_1, \dots, \underline{d}_q, \bar{d}_q\}$, such that

$$\forall s \in \{1, \dots, q\} \exists j \in \{1, \dots, p\} : \bar{d}_s \leq \tau_s^{\sigma_j}.$$

Then, the timed game automata TGA^{cl} is given by

$TGA^{cl} = (L^{cl}, \ell_0^{cl}, Act_c^{cl}, Act_u^{cl}, C^{cl}, E^{cl}, Inv^{cl})$ where

- $L^{cl} = \bigcup_{j=1}^p L^{\sigma_j} \cup \bigcup_{s=1}^q \{\mathcal{R}_s, Ear_s\};$
- $\ell_0^{cl} = \mathcal{R}_s$ such that $\xi(0) \in \mathcal{R}_s^{\sigma_1};$
- $Act_c^{cl} = Act^{\sigma_1} \cup \bigcup_{j=1}^p \{a_j^{cl}\};$
- $Act_u^{cl} = \{up!\};$
- $C^{cl} = C^{\sigma_1};$
- $E^{cl} = \bigcup_{s=1}^q \bigcup_{t \in \mathcal{E}_s} \{(Ear_s, c = 0, up!, \emptyset, \mathcal{R}_t)\} \cup \bigcup_{s=1}^q \bigcup_{j=1}^p \{(\mathcal{R}_s, c = 0, a_j^{cl}, \emptyset, \mathcal{R}_s^{\sigma_j}), (\mathcal{R}_s^{\sigma_j}, \underline{d}_s \leq c \leq \bar{d}_s, *, \{c\}, Ear_s)\} \cup \bigcup_{s=1}^q \bigcup_{j=1}^p \bigcup_{t \in \{t \mid (\mathcal{R}_s \rightarrow \mathcal{R}_t) \in E^{\sigma_j}\}} \{(\mathcal{R}_s^{\sigma_j}, \tau_s^{\sigma_j} \leq c \leq \bar{\tau}_s^{\sigma_j}, up!, \{c\}, \mathcal{R}_t)\};$

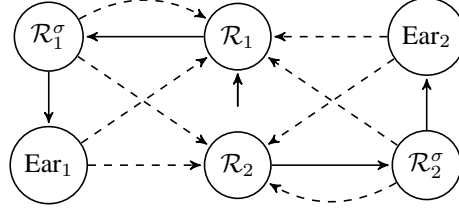


Figure 4: A timed game automaton modeling a control loop with an event-triggered implementation. The labels over edges and location invariants are not shown for simplicity.

- $\text{Inv}^{cl}(\mathcal{R}_s^{\sigma_j}) = \{c \mid c \leq \bar{\tau}_s^{\sigma_j}\}, \text{Inv}^{cl}(\mathcal{R}_s) = \{c \mid c = 0\},$
 $\text{Inv}^{cl}(\text{Ear}_s) = \{c \mid c = 0\}.$

In model just introduced, we use separate locations associated to each triggering coefficient and introduce the additional locations \mathcal{R}_s and Ear_s for $s \in \{1, \dots, q\}$. Both the locations \mathcal{R}_s and $\mathcal{R}_s^{\sigma_j}$ represent that the sampled state is in $\mathcal{R}_s^{\sigma_1}$. In location \mathcal{R}_s the scheduler has not chosen the triggering coefficient, whereas in location $\mathcal{R}_s^{\sigma_j}$ the scheduler has chosen triggering coefficient σ_j . Since the scheduler can choose the triggering coefficient, the edges from \mathcal{R}_s to $\mathcal{R}_s^{\sigma_j}$ are labeled with the controllable action a_j . After choosing the triggering coefficient, the scheduler is allowed to either: force earlier controller updates, or use the event-triggering mechanism (based on the chosen triggering coefficient).

If the scheduler decides to use the event-triggering mechanism, while staying in location $\mathcal{R}_s^{\sigma_j}$, the strategy is “do nothing”. This ensures that the outgoing edge to Ear_s is not taken. When the value of c is between $\underline{\tau}_s^{\sigma_j}$ and $\bar{\tau}_s^{\sigma_j}$, the event-triggering mechanism is activated. In this case, the edges from $\mathcal{R}_s^{\sigma_j}$ to \mathcal{R}_t labeled with the uncontrollable action $up!$ are enabled. Recall that the scheduler cannot choose the exact update time when using the event-triggering mechanism. This also implies that the scheduler cannot choose the region containing the next sampled state.

If the scheduler decides to force earlier controller updates, the scheduler will take the edge to Ear_s when that edge is enabled. In this case, the scheduler is able to choose the exact update time. Thus, the edges from $\mathcal{R}_s^{\sigma_j}$ to Ear_s are labeled with the controllable action $*$. In location Ear_s , the time cannot elapse and one of the outgoing edges has to be taken immediately. Since the scheduler cannot choose the region containing the next sampled state, the outgoing edges of Ear_s are labeled with the uncontrollable action $up!$. The outgoing edges are defined as follows: there exists an edge from Ear_s to \mathcal{R}_t if $t \in \mathcal{E}_s := \{t \mid \mathcal{X}_{[d_s, \bar{d}_s]}(\mathcal{R}_s^{\sigma_1}) \cap \mathcal{R}_t^{\sigma_1} \neq \emptyset\}$. A graphical representation of a TGA generated by Definition 9 is shown in Fig. 4.

In this subsection, we assume the initial conditions of the LTI system are a subset of a region. If the initial conditions are intersected with a set of regions $\mathcal{R}_{init} \subseteq \{\mathcal{R}_1^{\sigma_1}, \dots, \mathcal{R}_q^{\sigma_1}\}$, we can modify the TGA generated by Definition 9 as follows. Introduce a new location called \mathcal{R}_0 with invariant $\{c = 0\}$ and define

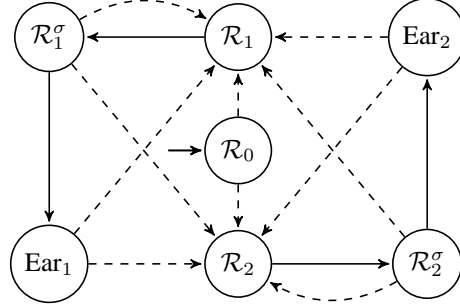


Figure 5: A timed game automaton modeling a control loop where the initial states are intersected with both conic regions. The labels over edges and location invariants are not shown for simplicity.

\mathcal{R}_0 as the initial location. Then, define edges from \mathcal{R}_0 to every location in \mathcal{R}_{init} with guard $c = 0$, action \otimes and without resetting the clock. Finally, action \otimes is defined as an uncontrollable action. In the above modification, the environment has to choose one of the locations corresponding to initial conditions when the system is started. A graphical representation of the TGA representing this situation is depicted in Fig. 5.

Proposition 1. *Switching between different triggering coefficients or triggering earlier does not hinder stability.*

Proof. Consider Lyapunov function $V : \mathbb{R}^n \rightarrow \mathbb{R}_{\geq 0}$ satisfying $\dot{V}(\xi(t)) \leq -\lambda_c V(\xi(t))$, with $\lambda_c > 0$ for the system (1) with continuous feedback $u(t) = K\xi(t)$. It has been shown in [12] that selecting a triggering coefficient $0 < \sigma < \bar{\sigma}$, with $\bar{\sigma}$ an appropriate constant depending on the LTI dynamics and the state-feedback gain, the event-triggered controller implementation (2)-(3) satisfies $\forall t : \dot{V}(\xi(t)) \leq -\lambda_e(\sigma)V(\xi(t))$, with $\lambda_c > \lambda_e(\sigma_1) > \lambda_e(\sigma_2) > 0$ for $0 < \sigma_1 < \sigma_2 < \bar{\sigma}$.

In fact, the triggering mechanism guarantees that $\dot{V}(\xi(t)) \leq -\lambda_e(\sigma)V(\xi(t))$ in the interval $t \in [t_k, t_k + \tau_\sigma(x)]$, for all $x = \xi(t_k)$. Since $\sigma_j < \bar{\sigma}$ for all $j \in \{1, \dots, p\}$, switching between different triggering coefficients guarantees that $\forall t : \dot{V}(\xi(t)) \leq -\lambda_e \tilde{\sigma} V(\xi(t))$, with $\tilde{\sigma} := \max_{j \in \{1, \dots, p\}} \sigma_j$.

Finally, if the system is forced to employ earlier triggering the assumption: $\forall s \in \{1, \dots, q\} \exists j \in \{1, \dots, p\} : \bar{d}_s \leq \tau_s^{\sigma_j}$ guarantees that the update occurs in the interval $[t_k, t_k + \tau_{\tilde{\sigma}}(x)]$, and thus $\forall t : \dot{V}(\xi(t)) \leq -\lambda_e \tilde{\sigma} V(\xi(t))$, which concludes the proof. \square

As it was mentioned in the beginning of Section 3.1, the NTGA associated with a set of NCSs, denoted by TGA^{NCSs} , is obtained by taking the parallel composition of the TGA associated with the network and with all control loops. In other words, $\text{TGA}^{NCSs} := \text{TGA}^{net} \mid \text{TGA}^{cl1} \mid \dots \mid \text{TGA}^{clN}$ where TGA^{cli} , $i \in \{1, \dots, N\}$, represents the TGA associated with the i -th control loop. The state of TGA^{NCSs} is described by a $(2N + 2)$ -tuple $(\ell_{net}, \ell_1, \dots, \ell_N, u_{net}, u_1, \dots, u_N)$ where ℓ_{net} is the location of TGA^{net} , ℓ_i is the location of TGA^{cli} , u_{net} is

clock assignment of TGA^{net} and u_i is the clock assignment of TGA^{cli} , for $i \in \{1, \dots, N\}$.

3.2 Specification: Safety

We are interested in finding a strategy such that the trajectories of the NTGA never enter the states corresponding to a conflict. Recall that a conflict corresponds to the following situation: a control loop is requesting updates when the communication network is busy. In our NTGA model, conflicts are captured by the active location of TGA^{net} becoming *Bad*. Thus the set of states we aim at avoiding \mathcal{A} contains all states such that the location of TGA^{net} is *Bad*, i.e.

$$\mathcal{A} = \{(\ell_{net}, \ell_1, \dots, \ell_N, u_{net}, u_1, \dots, u_N) \mid \ell_{net} = \text{Bad}\}.$$

3.3 Limiting the Consecutive Earlier Updates

If we allow the scheduler to force earlier controller updates, nothing prevents the scheduler from always using such type of updates and never employ the event-triggering mechanism. In this section, we discuss an approach to prevent the undesired schedule by limiting the consecutive earlier updates. In this approach, once a pre-specified limit has been reached, the scheduler is forced to use an event-triggering mechanism.

For this purpose we employ global integer variables, which are an extended feature of UPPAAL-Tiga modeling language to ease the modeling task but not part of the standard definition of TGA (cf. Definition 3). We define a global integer constant **earMax** representing the maximum consecutive earlier updates, and a global integer variable **earNum** to be used as a counter of consecutive earlier updates. A variable is global if it can be accessed by all TGAs. Finally, the resulting TGA is defined as follows.

Definition 10. Consider a set of timed automata $\text{TA}^{\sigma_j} = (L^{\sigma_j}, \ell_0^{\sigma_j}, \text{Act}^{\sigma_j}, C^{\sigma_j}, E^{\sigma_j}, \text{Inv}^{\sigma_j})$ generated from an event-triggered control loop with triggering coefficient $\sigma_j \in]0, \bar{\sigma}[$ for $j \in \{1, \dots, p\}$ and assume that $\mathcal{R}_s^{\sigma_1} = \dots = \mathcal{R}_s^{\sigma_p}$ for all $s \in \{1, \dots, q\}$. Consider also some constant **earMax** and a set of earlier update time parameters $\{\bar{d}_1, \bar{d}_1, \dots, \bar{d}_q, \bar{d}_q\}$, such that

$$\forall s \in \{1, \dots, q\} \exists j \in \{1, \dots, p\} : \bar{d}_s \leq \tau_s^{\sigma_j}.$$

Then, the timed game automata with options for earlier update, choice of triggering coefficients and limiting the consecutive earlier updates is given by $\text{TGA}^{clim} = (L^{clim}, \ell_0^{clim}, \text{Act}_c^{clim}, \text{Act}_u^{clim}, C^{clim}, E^{clim}, \text{Inv}^{clim})$ where

- $L^{clim} = \bigcup_{j=1}^p L^{\sigma_j} \cup \bigcup_{s=1}^q \{\mathcal{R}_s, \text{Ear}_s\};$
- $\ell_0^{clim} = \mathcal{R}_s$ such that $\xi(0) \in \mathcal{R}_s^{\sigma_1};$
- $\text{Act}_c^{clim} = \{up!\} \cup \text{Act}^{\sigma_1} \cup \bigcup_{j=1}^p \{a_j^{clim}\};$

- $\text{Act}_u^{clim} = \emptyset$;
- $C^{clim} = C^{\sigma_1}$;
- $E^{clim} = \bigcup_{s=1}^q \bigcup_{t \in \mathcal{E}_s} \{(Ear_s, c = 0, up!, \emptyset, \mathcal{R}_t)\} \cup \bigcup_{s=1}^q \bigcup_{j=1}^p \{(\mathcal{R}_s, c = 0, a_j^{clim}, \emptyset, \mathcal{R}_s^{\sigma_j}), (\mathcal{R}_s^{\sigma_j}, (\underline{d}_s \leq c \leq \bar{d}_s) \wedge (\text{earNum} < \text{earMax}), *, \{c\} \wedge (\text{earNum} := \text{earNum} + 1), Ear_s)\} \cup \bigcup_{s=1}^q \bigcup_{j=1}^p \bigcup_{\{t | (\mathcal{R}_s \rightarrow \mathcal{R}_t) \in E^{\sigma_j}\}} \{(\mathcal{R}_s^{\sigma_j}, \underline{\tau}_s^{\sigma_j} \leq c \leq \bar{\tau}_s^{\sigma_j}, up!, \{c\} \wedge (\text{earNum} := 0), \mathcal{R}_t)\}$;
- $\text{Inv}^{clim}(\mathcal{R}_s^{\sigma_j}) = \{c \mid c \leq \bar{\tau}_s^{\sigma_j}\}, \text{Inv}^{clim}(\mathcal{R}_s) = \{c \mid c = 0\}$.

There are two differences between Definition 10 and Definition 9. First, in the edges from $\mathcal{R}_s^{\sigma_j}$ to Ear_s , we add condition $\text{earNum} < \text{earMax}$ to the guard and add statement $\text{earNum} := \text{earNum} + 1$ to the reset. The additional condition is used to guarantee that the counter of consecutive earlier updates is always smaller than or equal to its maximum. The statement on the reset is used to increase the counter variable by one, once an earlier update happens. Recall that an earlier update happens when one of these edges is taken (cf. Section 3.1.2). Second, in the edges from $\mathcal{R}_s^{\sigma_j}$ to \mathcal{R}_t , we add the statement $\text{earNum} := 0$ to the reset. Recall that taking these edges represents the event-triggering mechanism is used (cf. Section 3.1.2). Thus the counter of consecutive earlier updates is reset to zero. Notice that the variable earNum takes values in $\{0, 1, \dots, \text{earMax}\}$.

Remark 2. *Note that with the presented implementation, either of the control loops may exhibit an arbitrary number of consecutive earlier triggerings. This is because the maximum number of consecutive earlier triggerings being a global counter. The counter is reset to zero whenever any of the control loops runs in event-triggered fashion. By employing more counters, one could easily generalize this idea to limit the number of consecutive earlier triggerings for each loop.*

After these modifications, the NTGA associated to the set of NCSs becomes $\text{TGA}^{NCSs} := \text{TGA}^{net} \mid \text{TGA}^{clim1} \mid \dots \mid \text{TGA}^{climN}$ where TGA^{climi} represents the TGA associated with the i -th control loop for $i \in \{1, \dots, N\}$. In this new NTGA, the state of TGA^{NCSs} is described by a $(2N + 3)$ -tuple $(\ell_{net}, \ell_1, \dots, \ell_N, u_{net}, u_1, \dots, u_N, \text{earNum})$ which includes the additional counter earNum . It follows that the bad states \mathcal{A} are now given by

$$\{(\ell_{net}, \ell_1, \dots, \ell_N, u_{net}, u_1, \dots, u_N, \text{earNum}) \mid \ell_{net} = \text{Bad}\}.$$

3.4 Scheduler Operation

Formally, a scheduler implements a strategy f , see Definition 5, for the NTGA TGA^{NCSs} . The strategy f is applied to TGA^{NCSs} providing, based on the run ρ of TGA^{NCSs} up to that time instant the controllable action $f(\rho)$ that guarantees the satisfaction of the desired specification.

This means in practice that after each discrete transition of the NTGA, i.e. every time a transmission is placed on the network, first the strategy chooses a

triggering coefficient. Then the strategy decides which control loop is updated and also its update mechanism: early or event triggered. After such a transition, and possibly after some time elapses, the environment chooses the conic region containing the next sampled state, which results in a discrete transition of the NTGA, and the procedure is repeated.

Example 1. *Let us illustrate the use of strategies on an example consisting of two control loops, two triggering coefficients $\{\sigma_1, \sigma_2\}$ and an option for earlier updates. The initial location of the first and second control loop is \mathcal{R}_1 and \mathcal{R}_2 , respectively. Initially the run of TGA^{NCSs} is*

$$\rho_0 = (\text{Idle}, \mathcal{R}_1, \mathcal{R}_2, 0, 0, 0).$$

After each update, the scheduler selects a triggering coefficient, according to the strategy f . Suppose that the scheduler chooses σ_2 for the first control loop, i.e. $f(\rho_0) = a_2^{cl1}$. The resulting run is

$$\rho_1 = \rho_0 \xrightarrow[\text{TS}]{a_2^{cl1}} (\text{Idle}, \mathcal{R}_1^{\sigma_2}, \mathcal{R}_2, 0, 0, 0).$$

If the scheduler chooses σ_1 for the second control loop, i.e. $f(\rho_1) = a_1^{cl2}$, the run becomes

$$\rho_2 = \rho_1 \xrightarrow[\text{TS}]{0} (\text{Idle}, \mathcal{R}_1^{\sigma_2}, \mathcal{R}_2, 0, 0, 0) \xrightarrow[\text{TS}]{a_1^{cl2}} (\text{Idle}, \mathcal{R}_1^{\sigma_2}, \mathcal{R}_2^{\sigma_1}, 0, 0, 0).$$

Then the scheduler follows the strategy to decide which control loop is updated and also its update mechanism: early or event triggered. Suppose that the strategy decides to update the first control loop earlier at time \underline{d}_1 . First, the scheduler delays the system

$$\rho_3 = \rho_2 \xrightarrow[\text{TS}]{\underline{d}_1} (\text{Idle}, \mathcal{R}_1^{\sigma_2}, \mathcal{R}_2^{\sigma_1}, \underline{d}_1, \underline{d}_1, \underline{d}_1).$$

*Then an earlier update is performed, i.e. $f(\rho_3) = *$. Notice that action $*$ is the internal action associated with the first control loop. We have run*

$$\rho_4 = \rho_3 \xrightarrow[\text{TS}]{*} (\text{Idle}, \text{Ear}_1, \mathcal{R}_2^{\sigma_1}, \underline{d}_1, 0, \underline{d}_1).$$

Since time cannot elapse in Ear_1 , the environment has to choose the conic region containing the next sampled state immediately. If the environment chooses \mathcal{R}_3 , the result is run

$$\rho_5 = \rho_4 \xrightarrow[\text{TS}]{0} (\text{Idle}, \text{Ear}_1, \mathcal{R}_2^{\sigma_1}, \underline{d}_1, 0, \underline{d}_1) \xrightarrow[\text{TS}]{up} (\text{InUse}, \mathcal{R}_3, \mathcal{R}_2^{\sigma_1}, 0, 0, \underline{d}_1).$$

Notice that TGA^{net} and the TGA associated with the first control loop move simultaneously via synchronizing action up . Input action $up?$ belongs to TGA^{net} , whereas output action $up!$ belongs to the TGA corresponding to the first control loop. Then the scheduler follows the strategy to select a triggering coefficient for

the first control loop, for example σ_1 , i.e. $f(\rho_5) = a_1^{cl1}$. The network is available again after Δ time units, resulting in the runs:

$$\begin{aligned} \rho_6 = \rho_5 &\xrightarrow[TS]{0} (InUse, \mathcal{R}_3, \mathcal{R}_2^{\sigma_1}, 0, 0, \underline{d}_1) \xrightarrow[TS]{a_1^{cl1}} \\ & (InUse, \mathcal{R}_3^{\sigma_1}, \mathcal{R}_2^{\sigma_1}, 0, 0, \underline{d}_1) \xrightarrow[TS]{\Delta} (InUse, \mathcal{R}_3^{\sigma_1}, \mathcal{R}_2^{\sigma_1}, \Delta, \Delta, \underline{d}_1 + \Delta), \end{aligned}$$

while the network is being used, and

$$\rho_7 = \rho_6 \xrightarrow[TS]{*} (Idle, \mathcal{R}_3^{\sigma_1}, \mathcal{R}_2^{\sigma_1}, \Delta, \Delta, \underline{d}_1 + \Delta)$$

once the network is released. Notice that action $*$ is the internal action associated with \mathbf{TGA}^{net} .

Note that this kind of scheduler is a centralized scheduler that needs to have a perfect overview of the transmissions placed on the network, and the control loop responsible for it. Furthermore, given that the locations of \mathbf{TGA}^{NCSs} are related to the actual sampled states transmitted through the network, the scheduler also needs to be able to read the content of the transmitted data.

4 Case Study

We showcase the results in an example comprising two event-triggered NCSs sharing the same communication network. The first control loop is given by [12, p. 1683]

$$\begin{aligned} \dot{\xi} &= \begin{bmatrix} 0 & 1 \\ -2 & 3 \end{bmatrix} \xi + \begin{bmatrix} 0 \\ 1 \end{bmatrix} v, \\ v &= \begin{bmatrix} 1 & -4 \end{bmatrix} \xi. \end{aligned} \tag{5}$$

The second control loop is given by [49, p. 1699]

$$\begin{aligned} \dot{\xi} &= \begin{bmatrix} -0.5 & 0 \\ 0 & 3.5 \end{bmatrix} \xi + \begin{bmatrix} 1 \\ 1 \end{bmatrix} v, \\ v &= \begin{bmatrix} 1.02 & -5.62 \end{bmatrix} \xi. \end{aligned} \tag{6}$$

In the sequel, we discuss two experimental results for the above example. Each experiment is characterized by four parameters: the number of conic regions q , the set of triggering coefficients $\{\sigma_1, \dots, \sigma_q\}$, the set of earlier update parameters $\{\underline{d}_1, \bar{d}_1, \dots, \underline{d}_q, \bar{d}_q\}$ and maximum consecutive earlier triggering \mathbf{earMax} .

4.1 Limiting the Consecutive Earlier Updates

In this experiment, the minimum channel occupancy time is $\Delta = 0.005$, the number of conic regions is $q = 200$ and there is one triggering coefficient $\sigma_1 = 0.05$. The input value can be updated 0.005 time units before the lower bound

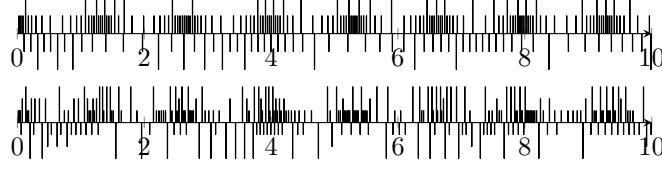


Figure 6: Status of the shared communication network up to 10 time units. The bars on the top and on the bottom of the x axis represent the network is being used by (6) and (5), respectively. The top and bottom plots represent the result of experiments discussed in Sections 4.1 and 4.2, respectively.

in all regions, i.e. $\underline{d}_s = \tau_s - 0.005$ and $\bar{d}_s = \tau_s$ for all $s \in \{1, \dots, q\}$. The maximum consecutive earlier triggering is **earMax** = 4.

We create a model in UPPAAL-Tiga according to Definition 10: the TGA for (5), (6) and the shared communication network are denoted **tgaT**, **tgaH** and **net**, respectively. The specification is given by

```
control: A[] not( net.Bad )
```

A strategy is generated with UPPAAL-Tiga to satisfy this specification. To illustrate the type of strategies synthesized, we show in the following a fragment of the strategy generated by UPPAAL-Tiga for the situation in which the locations of **tgaT**, **tgaH** and **net** are $\mathcal{R}_1^{\sigma_1}$, $\mathcal{R}_1^{\sigma_1}$ and *Idle*, respectively.

```
State: ( tgaT.R1a1 tgaH.R1a1 net.Idle )
earNum=3
When you are in (25<=tgaH.c && tgaT.c<65 && tgaH.c-tgaT.c<=-35)
  || (85<tgaT.c && 25<=tgaH.c && tgaT.c<105 && tgaH.c<=30)
  || (38<tgaT.c && 25<=tgaH.c && tgaT.c-tgaH.c<=30 && tgaH.c<=30)
  || (25<=tgaH.c && tgaT.c<31 && tgaH.c-tgaT.c<=-5)
  || (25<=tgaH.c && tgaT.c-tgaH.c<=-5 && tgaH.c<=30),
take transition tgaH.R1a1->tgaH.Ear1
{ c >= 25 && c <= 30 && earNum < earMax, up!, 1 }
net.Idle->net.InUse { 1, up?, c := 0 }
When you are in (105<=tgaT.c && tgaT.c <=111 && tgaH.c<25),
take transition tgaT.R1a1->tgaT.Ear1
{ c >= 105 && c <= 111 && earNum < earMax, up!, 1 }
net.Idle->net.InUse { 1, up?, c := 0 }
```

As shown above, two different conditions, based on the clock values of **tgaT**, clock values of **tgaH** and the difference of clock values in **tgaT** and **tgaH**, can be appreciated: if the first one is satisfied, an early update is forced for **tgaH** where the inter-sample time is between 25 and 30; if the second condition is satisfied, an early update is forced for **tgaT** where the inter-sample time is between 105 and 111. If none of the conditions are satisfied, no early update is forced, i.e. the strategy is to let time elapses for both loops.

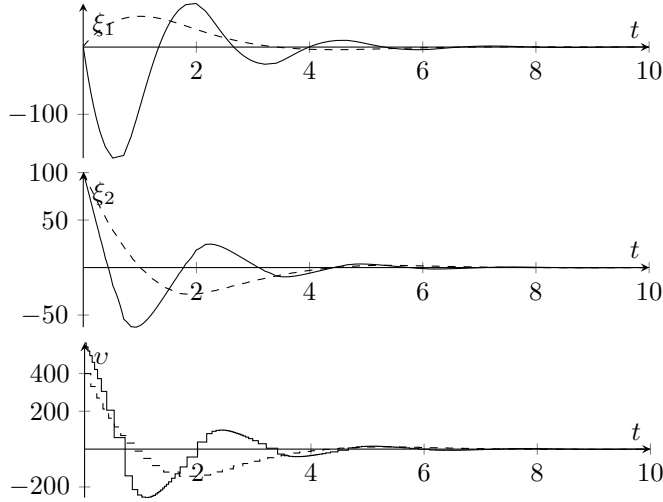


Figure 7: The state and input trajectories for the experiment in Section 4.1. The solid and dashed lines are the trajectories of (6) and (5), respectively.

The strategy generated by UPPAAL-Tiga was applied to the two NCSs (5)-(6), with both systems initialized at the state $[1, 100]^T$, corresponding to \mathcal{R}_1 in both of the timing abstractions for the systems. The network status is shown in Fig. 6 (top), where long and short bars represent event-triggered and earlier update mechanisms, respectively. Note that while either of the control loops may exhibit an arbitrary number of consecutive triggerings, the maximum number of consecutive earlier triggerings is respected to be below 4 as this counter is a shared (global) one that is reset to zero whenever any of the two loops run in event-triggered fashion. During the time horizon of 10, the input of (5) is updated 63 times consisting of 50 earlier updates and 13 event-triggering mechanisms. For (6), the input is updated 152 times consisting of 121 earlier updates and 31 event-triggering mechanisms. The state and input trajectories are shown in Fig. 7.

4.2 Choice of Triggering Coefficients

In this experiment, the minimum channel occupancy time is $\Delta = 0.005$, the number of conic regions is $q = 200$ and there are three triggering coefficients $\sigma_1 = 0.01$, $\sigma_2 = 0.03$ and $\sigma_3 = 0.09$.

We create again a model in UPPAAL-Tiga according to Definition 9: the TGA for (5), (6) and the shared communication network are denoted **tgaT**, **tgaH** and **net**, respectively. The specification is the same as in Section 4.1 and again we generate a strategy using UPPAAL-Tiga. The following is a fragment of the strategy generated by UPPAAL-Tiga when the location of **tgaT**, **tgaH** and **net** is \mathcal{R}_{37} , $\mathcal{R}_{38}^{\sigma_1}$ and *Idle*, respectively.

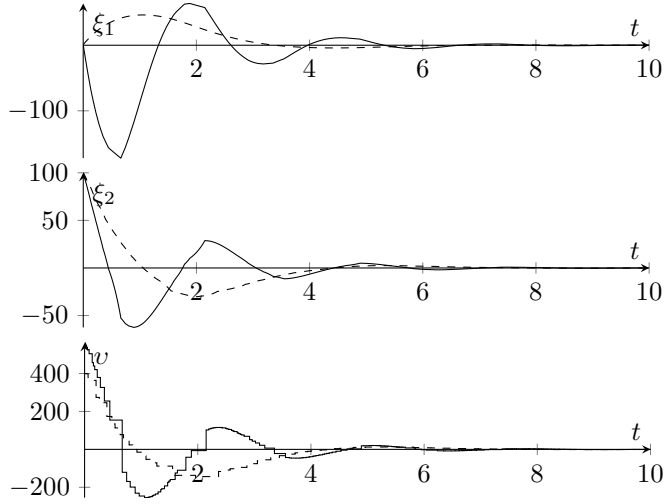


Figure 8: The state and input trajectories for the experiment in Section 4.2. The solid and dashed lines are the trajectories of (6) and (5), respectively.

```

State: ( tgaT.R37 tgaH.R38a1 net.Idle )
When you are in (tgaT.c==0 && 145<tgaH.c && tgaH.c<=154),
take transition tgaT.R37->tgaT.R37a1 { c == 0, tau, 1 }
When you are in (tgaT.c==0 && 65<=tgaH.c && tgaH.c<=102),
take transition tgaT.R37->tgaT.R37a2 { c == 0, tau, 1 }
When you are in (tgaT.c==0 && 102<tgaH.c && tgaH.c<=145)
|| (tgaT.c==0 && 5<=tgaH.c && tgaH.c<65),
take transition tgaT.R37->tgaT.R37a3 { c == 0, tau, 1 }

```

Notice that now there are three conditions: if the i -th condition is satisfied, the location of **tgaT** is forced to transit to $\mathcal{R}_{37}^{\sigma_i}$ for $i \in \{1, 2, 3\}$.

The strategy generated by UPPAAL-Tiga is applied to the NCSs (5)-(6), both with the state initialized at $[1, 100]^T$, corresponding in both cases with the initial location is \mathcal{R}_1 . The network status is shown in Fig. 6 (bottom). Short, medium and long bars represent event-triggered with triggering coefficient equals σ_1 , σ_2 and σ_3 , respectively. During the first 10 time units, the input of (5) is updated 84 times consisting of 56 updates using σ_1 , 6 updates using σ_2 and 22 updates using σ_3 . For (6), the input is updated 182 times consisting of 106 updates using σ_1 , 26 updates using σ_2 and 50 updates using σ_3 . The state and input trajectories are shown in Fig. 8.

5 Discussion and Future Work

We have provided an approach to synthesize conflict-free scheduling policies for sets of networked control systems (NCSs) with the possibilities of updating the

input value according to an event-triggering mechanism, selectable from a set of them, or earlier than the time dictated by such an event-triggering rule. As indicated in Section 3.3 the main limitations of this proposed scheduling scheme is its centralized nature, and the fact that the scheduler needs to be able to read the content of the messages sent through the network. The approach is nonetheless applicable to many setups encountered in practice in which different control systems are interconnected through a bus, e.g. CAN, EtherCAT or FlexRay. In such systems, every element connected to the bus can see the traffic flowing through the network. While it may be the case that the precise content of messages is not available (e.g. for potential security reasons), it is worth noting that for the scheduler only the abstracted state, i.e. the region \mathcal{R}_s , is relevant. Therefore we can envisage implementations or practical applications in which this sort of scheduling could be easily adopted.

In wireless settings, the event-triggered paradigm offers great benefits for energy consumption reduction, but network topologies can be in general more complex than a simple bus type of configuration. Therefore, interesting extensions of this work to allow decentralized scheduling, possibly including network topological constraints, would enable broader applicability of these techniques. Current and future work is focusing on these issues, extensions of the abstraction of the timing of event-triggered systems beyond LTI systems with state-feedback, and on the implementation of a tool-box automating the whole timing abstraction and scheduler synthesis proposed in [27] and the current paper respectively.

References

- [1] J. Liu, A. Gusrialdi, D. Obradovic, and S. Hirche, “Study on the effect of time delay on the performance of distributed power grids with networked cooperative control,” in *Proceedings of the 1st IFAC Workshop on Estimation and Control of Networked Systems*, 2009, pp. 168–173.
- [2] H. A. Thompson, “Wireless and internet communications technologies for monitoring and control,” *Control Engineering Practice*, vol. 12, no. 6, pp. 781–791, 2004.
- [3] D. Lehmann and J. Lunze, “Extension and experimental evaluation of an event-based state-feedback approach,” *Control Engineering Practice*, vol. 19, no. 2, pp. 101–112, 2011.
- [4] P. Antsaklis and J. Baillieul, “Special issue on technology of networked control systems,” *Proceedings of the IEEE*, vol. 95, no. 1, pp. 5–8, Jan. 2007.
- [5] G. Nair, F. Fagnani, S. Zampieri, and R. Evans, “Feedback control under data rate constraints: An overview,” *Proceedings of the IEEE*, vol. 95, no. 1, pp. 108–137, Jan. 2007.

- [6] J. P. Hespanha, P. Naghshtabrizi, and Y. Xu, “A survey of recent results in networked control systems,” *Proceedings of the IEEE*, vol. 95, pp. 138–162, Jan. 2007.
- [7] D. Hristu-Varsakelis and W. S. Levine, Eds., *Handbook of networked and embedded control systems*, ser. Control Engineering. Boston, MA: Birkhäuser Boston Inc., 2005.
- [8] K.-E. Årzén, A. Bicchi, S. Hailes, K. Johansson, and J. Lygeros, “On the design and control of wireless networked embedded systems,” in *IEEE Computer Aided Control System Design*, Oct. 2006, pp. 440–445.
- [9] M. Rabi and K. H. Johansson, “Event-triggered strategies for industrial control over wireless networks,” in *Proc. 4th Annual Int. Conf. Wireless Internet*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, p. 34.
- [10] M. Mazo Jr. and P. Tabuada, “Decentralized event-triggered control over wireless sensor/actuator networks,” *IEEE Transactions on Automatic Control*, *Special issue on Wireless Sensor Actuator Networks*, vol. 56, no. 10, pp. 2456–2461, Oct. 2011.
- [11] K. Åström and B. Bernhardsson, “Comparison of Riemann and Lebesgue sampling for first order stochastic systems,” in *Proceedings of the 41th IEEE Conference on Decision and Control (CDC’02)*, vol. 2, Dec. 2002, pp. 2011–2016.
- [12] P. Tabuada, “Event-triggered real-time scheduling of stabilizing control tasks,” *IEEE Transactions on Automatic Control*, vol. 52, no. 9, pp. 1680–1685, Sep. 2007.
- [13] M. Velasco, J. Fuertes, and P. Marti, “The self triggered task model for real-time control systems,” in *Proceedings of the 24th IEEE Real-Time Systems Symposium (Work in Progress)*, 2003, pp. 67–70.
- [14] W. Heemels, J. Sandee, and P. van den Bosch, “Analysis of event-driven controllers for linear systems,” *International Journal of Control*, vol. 81, no. 4, pp. 571–590, 2008.
- [15] A. Anta and P. Tabuada, “To sample or not to sample: Self-triggered control for nonlinear systems,” *IEEE Transactions on Automatic Control*, vol. 55, pp. 2030–2042, Sep. 2010.
- [16] M. Mazo Jr., A. Anta, and P. Tabuada, “An ISS self-triggered implementation of linear controller,” *Automatica*, vol. 46, pp. 1310–1314, Aug. 2010.
- [17] B. Sprunt, L. Sha, and J. Lehoczky, “Aperiodic task scheduling for hard-real-time systems,” *Real-Time Systems*, vol. 1, no. 1, pp. 27–60, 1989.

- [18] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer, 2011, vol. 24.
- [19] G. C. Walsh and H. Ye, “Scheduling of Networked Control Systems,” *IEEE Control Systems Magazine*, 2001.
- [20] A. Cervin and T. Henningsson, “Scheduling of event-triggered controllers on a shared network,” in *Proceedings of the 47th IEEE Conference on Decision and Control (CDC’08)*, Dec. 2008, pp. 3601–3606.
- [21] S. Al-Areqi, D. Gorges, S. Reimann, and S. Liu, “Event-based control and scheduling codesign of networked embedded control systems,” in *Proceedings of the 32nd American Control Conference (ACC’13)*, Jun. 2013, pp. 5299–5304.
- [22] S. Al-Areqi, D. Gorges, and S. Liu, “Stochastic event-based control and scheduling of large-scale networked control systems,” in *Proceedings of the European Control Conference*, Jun. 2014, pp. 2316–2321.
- [23] S. Reimann, S. Al-Areqi, and S. Liu, “An event-based online scheduling approach for networked embedded control systems,” in *Proceedings of the 32nd American Control Conference (ACC’13)*, Jun. 2013, pp. 5326–5331.
- [24] P. Bouyer, F. Cassez, E. Fleury, and K. Larsen, “Optimal strategies in priced timed game automata,” in *Foundations of Software Technology and Theoretical Computer Science (FSTTCS’04)*, ser. Lecture Notes in Computer Science, K. Lodaya and M. Mahajan, Eds. Springer, Heidelberg, 2005, vol. 3328, pp. 148–160.
- [25] —, “Synthesis of optimal strategies using HYTECH,” *Electronic Notes in Theoretical Computer Science*, vol. 119, no. 1, pp. 11–31, 2005.
- [26] O. Maler, A. Pnueli, and J. Sifakis, “On the synthesis of discrete controllers for timed systems,” in *Proc. 12th Symp. on Theoretical Aspects of Computer Science (STACS’95)*, ser. Lecture Notes in Computer Science, E. Mayr and C. Puech, Eds. Springer, Heidelberg, 1995, vol. 900, pp. 229–242.
- [27] A. Sharifi Kolarijani and M. Mazo Jr., “A formal traffic characterization of LTI event-triggered control systems,” *CoRR*, vol. abs/1503.05816, 2015.
- [28] L. De Alfaro, T. Henzinger, and R. Majumdar, “Symbolic algorithms for infinite-state games,” in *Concurrency Theory (CONCUR’01)*, ser. Lecture Notes in Computer Science, K. Larsen and M. Nielsen, Eds. Springer, Heidelberg, 2001, vol. 2154, pp. 536–550.
- [29] F. Cassez, A. David, E. Fleury, K. Larsen, and D. Lime, “Efficient on-the-fly algorithms for the analysis of timed games,” in *Concurrency Theory (CONCUR’05)*, ser. Lecture Notes in Computer Science, M. Abadi and L. de Alfaro, Eds. Springer, Heidelberg, 2005, vol. 3653, pp. 66–80.

- [30] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis, “Controller synthesis for timed automata,” in *Proc. IFAC Symp. on System Structure & Control*, 1998, pp. 469–474.
- [31] R. Alur and D. Dill, “A theory of timed automata,” *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.
- [32] A. Ravn, J. Srba, and S. Viglio, “Modelling and verification of web services business activity protocol,” in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS’11)*, ser. Lecture Notes in Computer Science, P. Abdulla and K. Leino, Eds. Springer, Heidelberg, 2011, vol. 6605, pp. 357–371.
- [33] K. Havelund, A. Skou, K. Larsen, and K. Lund, “Formal modeling and analysis of an audio/video protocol: an industrial case study using UP-PAAL,” in *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS’97)*, Dec. 1997, pp. 2–13.
- [34] P. D’Argenio, J.-P. Katoen, T. Ruys, and J. Tretmans, “The bounded retransmission protocol must be on time!” in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS’97)*, ser. Lecture Notes in Computer Science, E. Brinksma, Ed. Springer, Heidelberg, 1997, vol. 1217, pp. 416–431.
- [35] L. Aceto, A. Burgueño, and K. Larsen, “Model checking via reachability testing for timed automata,” in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS’98)*, ser. Lecture Notes in Computer Science, B. Steffen, Ed. Springer, Heidelberg, 1998, vol. 1384, pp. 263–280.
- [36] H. Jensen, K. Larsen, and A. Skou, “Modelling and analysis of a collision avoidance protocol using SPIN and UPPAAL,” *BRICS Report Series*, vol. 3, no. 24, 1996.
- [37] A. David and W. Yi, “Modelling and analysis of a commercial field bus protocol,” in *12th Euromicro Conference on Real-Time Systems*, 2000, pp. 165–172.
- [38] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, “Symbolic model checking for real-time systems,” *Information and Computation*, vol. 111, no. 2, pp. 193–244, 1994.
- [39] A. Fehnker, “Scheduling a steel plant with timed automata,” in *Proc. 6th Int. Conf. Real-Time Computing Systems and Applications (RTCSA’99)*, 1999, pp. 280–286.
- [40] Y. Abdeddaïm and O. Maler, “Job-shop scheduling using timed automata?” in *Computer Aided Verification (CAV’01)*, ser. Lecture Notes in Computer Science, G. Berry, H. Comon, and A. Finkel, Eds. Springer, Heidelberg, 2001, vol. 2102, pp. 478–492.

- [41] Y. Abdeddaïm, A. Kerbaa, and O. Maler, “Task graph scheduling using timed automata,” in *Proc. Int. Parallel and Distributed Processing Symposium (IPDPS’03)*, Apr. 2003, pp. 8 pp.–.
- [42] G. Behrmann, E. Brinksma, M. Hendriks, and A. Mader, “Scheduling lacquer production by reachability analysis – a case study,” in *Proceedings of the 16th IFAC World Congress*. Elsevier, 2005.
- [43] —, “Production scheduling by reachability analysis - a case study,” in *Proc. 19th IEEE Int. Parallel and Distributed Processing Symposium (IPDPS’05)*, Apr. 2005, pp. 140a–140a.
- [44] G. Behrmann, A. Fehnker, T. Hune, K. Larsen, P. Pettersson, J. Romijn, and F. Vaandrager, “Minimum-cost reachability for priced time automata,” in *Hybrid Systems: Computation and Control (HSCC’01)*, ser. Lecture Notes in Computer Science, M. Di Benedetto and A. Sangiovanni-Vincentelli, Eds. Springer, Heidelberg, 2001, vol. 2034, pp. 147–161.
- [45] R. Alur, S. La Torre, and G. Pappas, “Optimal paths in weighted timed automata,” in *Hybrid Systems: Computation and Control (HSCC’01)*, ser. Lecture Notes in Computer Science, M. Di Benedetto and A. Sangiovanni-Vincentelli, Eds. Springer, Heidelberg, 2001, vol. 2034, pp. 49–62.
- [46] P. Bouyer, E. Brinksma, and K. Larsen, “Optimal infinite scheduling for multi-priced timed automata,” *Formal Methods in System Design*, vol. 32, no. 1, pp. 3–23, 2008.
- [47] J. Bengtsson and W. Yi, “Timed automata: Semantics, algorithms and tools,” in *Lectures on Concurrency and Petri Nets*, ser. Lecture Notes in Computer Science, J. Desel, W. Reisig, and G. Rozenberg, Eds. Springer, Heidelberg, 2004, vol. 3098, pp. 87–124.
- [48] G. Behrmann, A. David, and K. Larsen, “A tutorial on UPPAAL,” in *Formal Methods for the Design of Real-Time Systems (SFM-RT’04)*, ser. Lecture Notes in Computer Science, M. Bernardo and F. Corradini, Eds., vol. 3185. Springer, Heidelberg, Sep. 2004, pp. 200–236.
- [49] L. Hetel, A. Kruszewski, W. Perruquetti, and J.-P. Richard, “Discrete and intersample analysis of systems with aperiodic sampling,” *IEEE Transactions on Automatic Control*, vol. 56, no. 7, pp. 1696–1701, Jul. 2011.